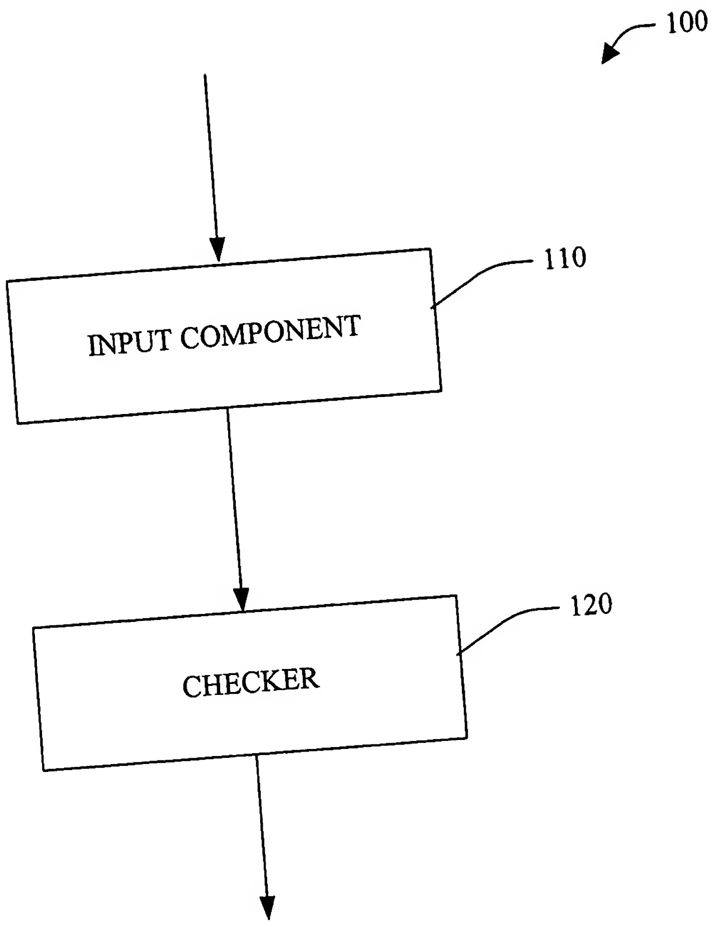


FIG. 1

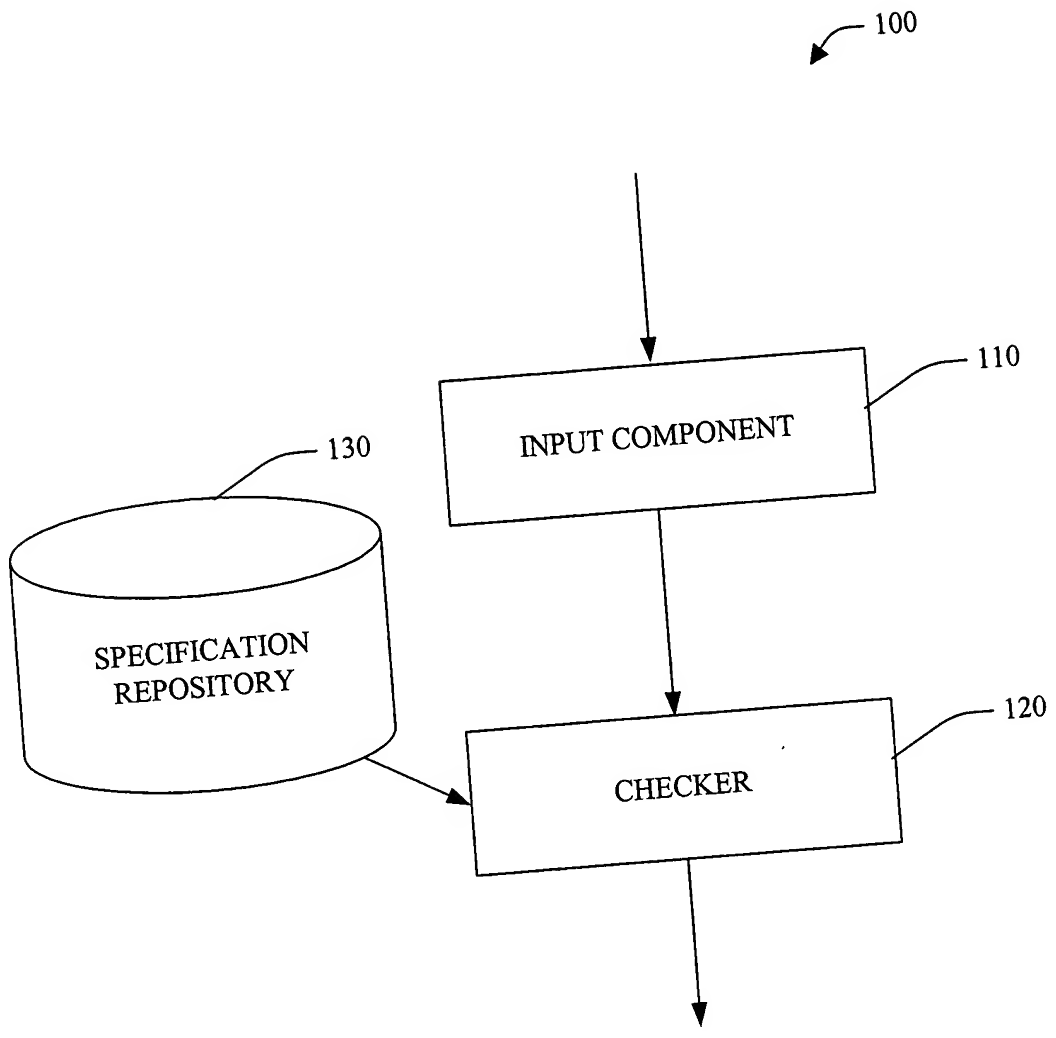

100
INPUT COMPONENT
110
130
SPECIFICATION REPOSITORY
CHECKER
120

FIG. 2

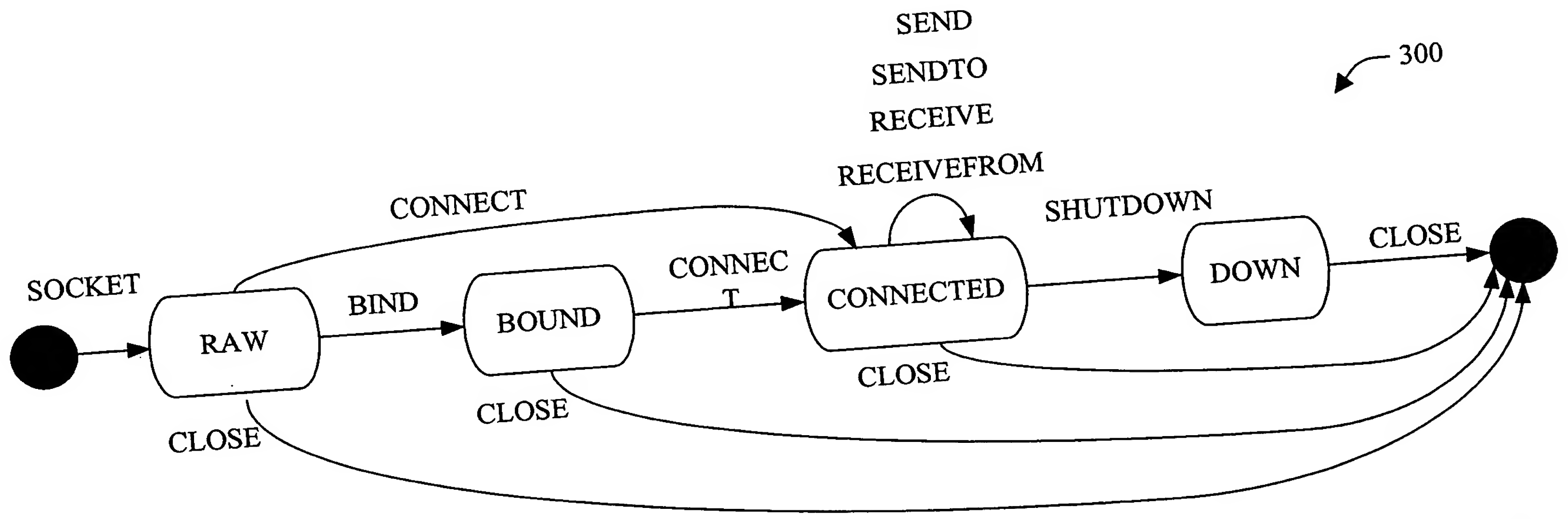

SEND
SENDTO
RECEIVE
RECEIVEFROM
300
CONNECT
SOCKET
RAW
BIND
BOUND
CONNEC
T
CONNECTED
SHUTDOWN
DOWN
CLOSE
CLOSE
CLOSE
CLOSE

FIG. 3

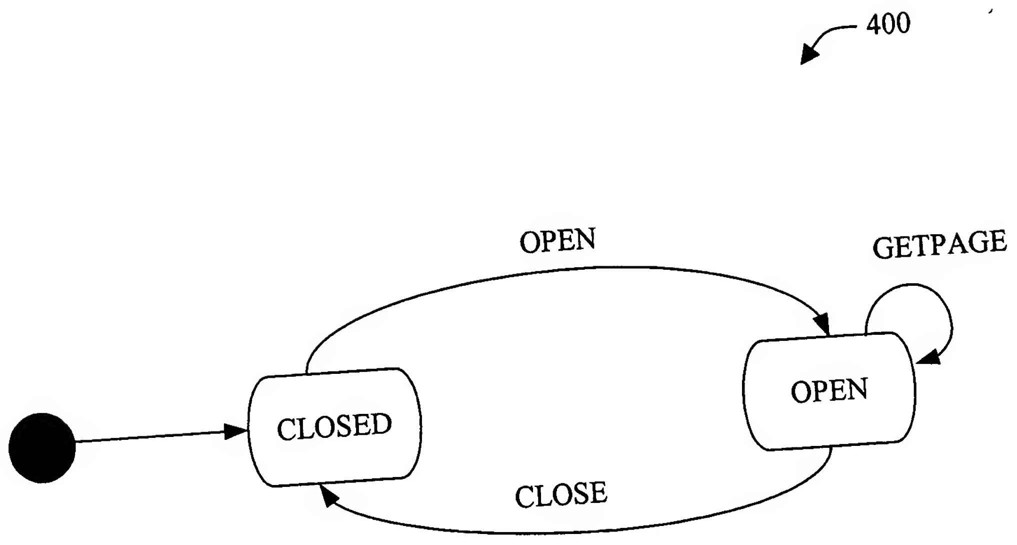


FIG. 4

500

```
[WithProtocol(
        CustomState=typeof(SqlConnectionState)) ]
class SqlConnection
{
[ Creates,
 OutConnectionState(
         Status=ConnectionState.Closed,
         Host="", Database="")]
 SqlConnection ();

 [ Creates,
  OutConnectionState(
          Status=ConnectionState.Closed,
          StateProvider="NewHostAndDatabase"),
  OutStateDependsOn("cennectionString")]
  SqlConnection (string connectionString) ;

  [ OutConnectionState(
           Status=ConnectionState.Open) ]
  void Open () ;
  }
```

FIG. 5

600

```
[ WithProtocol(
        CustomStat=typeof(SqlCommandState)) ]
class SqlCommand
{
  [OutCommandState(
        StateProvider="UpdateCommandText"),
   OutStateDependsOn("cmdText") ]
  SqlCommand (string cmdText);

  [ property: Transparent ]
  SqlConnection Connection { get; set; }

  [ InCommandState(
        StateChecker="CheckCommandText"),
   InStateDependdsOn("this.Connection") ]
  [ return: OutReaderState(
        StateProvider="GetColumnInfo"),
   OutStateDependsOn("this.Connection","this") ]
  SqlDataReader ExecuteReader ();
}
```

FIG. 6

700

```
{ WithProtocol(
        CustonState=typeof(sqlReaderState)) ]
class SqlDataReader
{
        [ InReaderState(
          StateChecker="ValidColumnName"),
          InStateDependsOn("name") ]
        object get_Item (string name);

        [ InReaderState(
          StateChecker="ColumnIsString"),
          InStateDependsOn("i") ]
        string Getstring (int i);
  }
```

FIG. 7

800

```
class SqlConnectionState : CustomState
{
          ConnectionState Status
          sting Host, Database;

          void NewHostAndDatabase (string{} connString) {
          // Example plug-in postcondition, which
          // parses a connection string for
          // its host and database names.
          Regex hostRegex = new Regex (
                    @"(data source|server)\s*=([^;]*)\b",
                    RegexOptions.IgnoreCase);
          Regex dbRegex = new Regex(
                    @"(catalog|database)\s*=([^;}*\b",
                    RegexOptions.IgnoreCase);
          for (int i=0; i<connString.Length; i++) {
                    MatchCollection dbm =
                      hostRegex.Matches(connString[i]);
                    if (dbm.Count > 0)
                       Host = dbm[0] .Groups[2].Captures[0].Value;
                    MatchCollection hm =
                       dbRegex.Matches(connString[i]);
                    if (hm.Count > 0)
                       Database = hm[0].Groups[2].Captures[0].Value;

          }
          if (Host == null)
                    Fail("could not find host");
          if (Database == null)
                    Fail("could not find database");

          }
}
```

FIG. 8

900

```
class SqlCommandState : CustomState
{
        string[] CommandText;

        void UpdateCommandText (string[] c0 { CommandText=c; }

        bool CheckCommandText (SqlConnectionState c) {
            return ISLegalSQL(CommandText, c.Host, c.Database);
        }
}
```

FIG. 9

1000

```
class SqlReaderState : CustomState
{
        string [] ColumnNames, ColumnTypes;

        void GetColumnInfor (SqlConnectionState connection,
                                SqlComandState command) {...}
        bool ValidColumnName (string[] name) {...}
        bool ClumnIsString (int i) {...}
}
```

FIG. 10

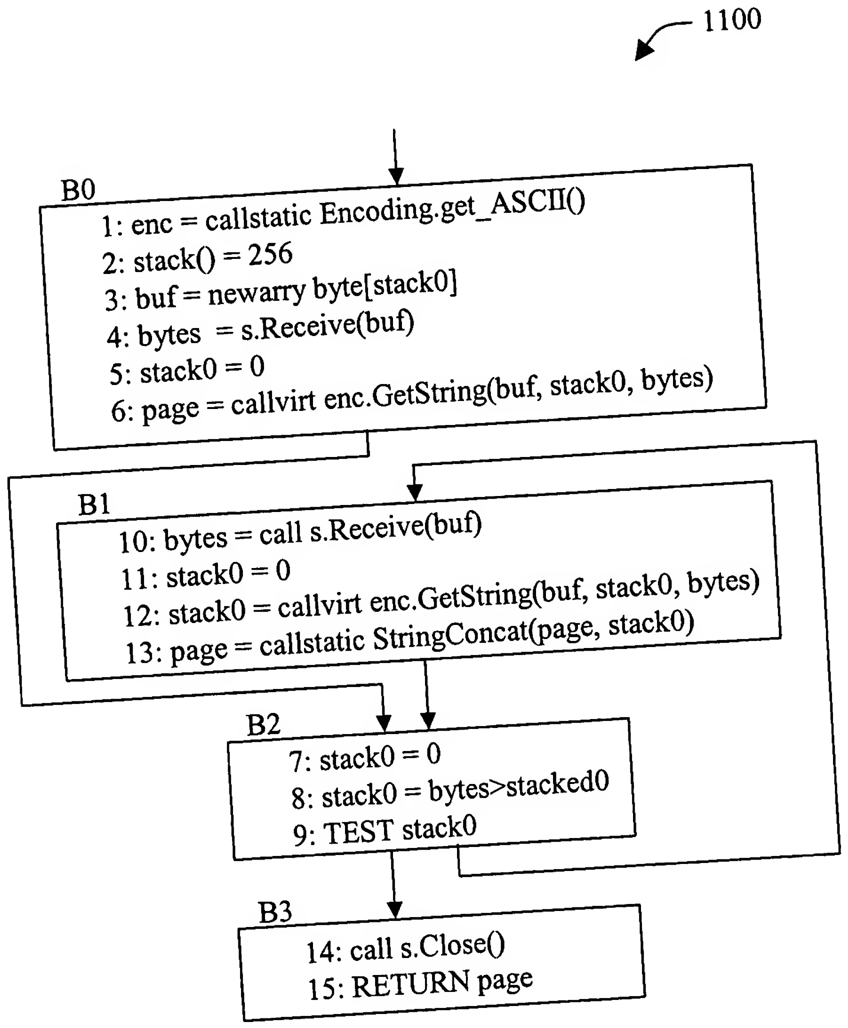

1100
B0
1: enc = callstatic Encoding.get_ASCII()
2: stack() = 256
3: buf = newarry byte[stack0]
4: bytes = s.Receive(buf)
5: stack0 = 0
6: page = callvirt enc.GetString(buf, stack0, bytes)
B1
10: bytes = call s.Receive(buf)
11: stack0 = 0
12: stack0 = callvirt enc.GetString(buf, stack0, bytes)
13: page = callstatic StringConcat(page, stack0)
B2
7: stack0 = 0
8: stack0 = bytes>stacked0
9: TEST stack0
B3
14: call s.Close()
15: RETURN page

FIG. 11

1200

0	s : ref(a_0)	a_0 → (Socket.NotAliased,"connected", 0)
1	enç : ref(a_1)	a_1 → (Encoding.MayBeAliased/Escaping, default, 0)
2	stack0 : calue(int, 256, default)	
3	buf : value(byte[],·, default)	
4	bytes : value(int, ·, default)	
5	stack0 : value(int, 0, default)	a_3 → (string.MayBeAliased/Escaping,default, 0)
6	page : ref(a_3)	
7	stack0 : value(int, 0, default)	
8	stack0 : value(bool, ·, default)	
9	(no change)	
10	bytes : value(int, ·, default)	
11	stack0 : value(int, 0, default)	a_4 → (string.MayBeAliased/Escaping,default, 0)
12	stack0 : ref(a_4)	
13	(no change)	(a_0 removed from capabilities)
14		
15	(no change)	

FIG. 12

1300

this : ref (a_0)
a_0 → (WebPageFetcher.NA, "open", 0)
1: stack0 = this.socket
this : ref(a_0)
stack0 : ref(a_1)
a_0 → (WebPageFetcher, NA, "open". {socket → a_1})
a_1 → (Socket, NA, "connected", 0)
2: stack1 = callstatic Encoding.get_ASCII()
this : ref (a_0)
stack0 : ref(a_1)
a_0 →(WebPageFetcher, NA, "open". {socket → a_1})
a_1 → (Socket, NA, "connected", 0)
a_2 → (Encoding, MA/E,default, 0)
3: stack2 - "Quit\n"
this : ref(a_0)
stack0 : ref(a_1)
stack0 : ref(a_2)
stack2 : value(string, "QUIT", default)
a_0 → (WebPageFetcher, NA, "open". {socket → a_1})
a_1 → (Socket, NA, "connected", 0)
a_2 → (Encoding, MA/E,default, 0)
4: stack1 = callvirt stack1. GetBytes(stack2)
this : ref (a_0)
stack0 : ref(a_1)
stack1 : ref(a_3)
a_0 → (WebPageFetcher, NA, "open". {socket → a_1})
a_1 → (Socket, NA, "connected", 0)
a_2 → (Encoding, MA/E,default, 0)
a_3 → (byte[],MA/E, default, 0)
5: stack0 = call stack0.Send (stack1)
this : ref (a_0)
stack1 : ref(a_3)
a_0 → (WebPageFetcher, NA, "open". {socket → a_1})
a_1 → (Socket, NA, "connected", 0)
a_2 → (Encoding, MA/E,default, 0)
a_3 → (byte[],MA/E, default, 0)

FIG. 13A

1300

```
6:    stack0 = this.socket
      this : ref (a0)
      stack0 : ref(a1)
      stack1 : ref(a3)
      a0 g (WebPageFetcher, NA, "open". {socket g a1})
      a1 g (Socket, NA, "connected", 0)
      a2 g (Encoding, MA/E,default, 0)
      a3 g (byte[],MA/E, default, 0)
7:    call stack0.Close()
      this : ref (a0)
      stack0 : ref(a1)
      stack1 : ref(a3)
      a0 g (WebPageFetcher, NA, "open". {socket g a1})
      a2 g (Encoding, MA/E,default, 0)
      a3 g (byte[],MA/E, default, 0)
8:    return
      this : ref (a0)
      a0 g (WebPageFetcher, NA, "open". {socket g a1})
```

FIG. 13B

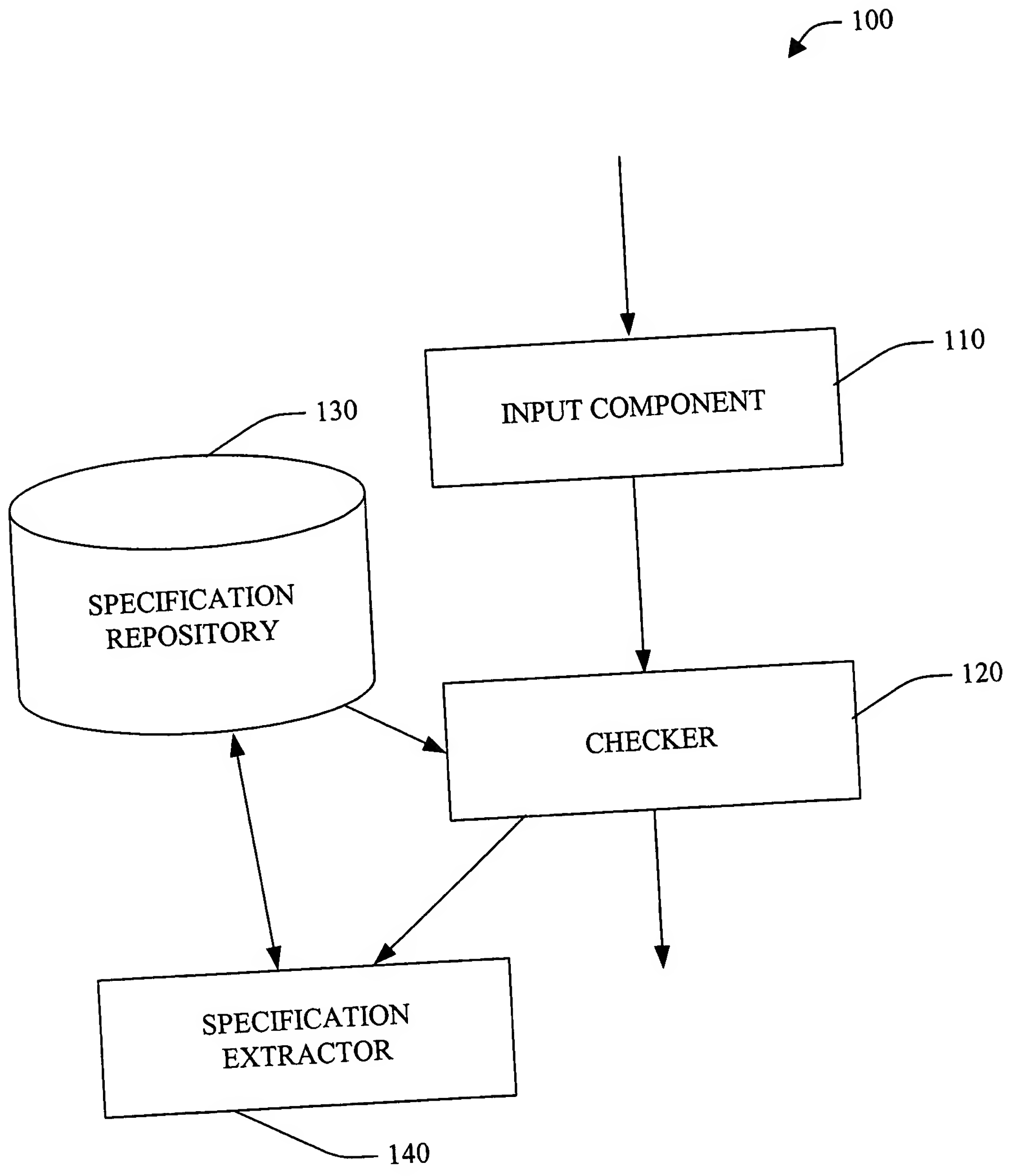

100
INPUT COMPONENT
110
130
SPECIFICATION REPOSITORY
CHECKER
120
SPECIFICATION EXTRACTOR
140

FIG. 14

1500

```
[WithProtocol( UnknownDB, KnownDB)]
class Publications : System.Web.UI.Page
{
  [InConnectionState(WhenEnclosingState=UnknownDB
    Status = ConnectionState.Closed,
    Host = AnyHost, Database = AnyDatabase)
   InConnectionState(WhenEnclosingState=KnownDB
     Status = ConnectionState.Closed,
     Host = XXX, Database = YYY )
  private SqlConnection m_sqlCn;

  [ChangesState( UnknownDB , KnownDB )]
  private void OnPageLoad (EventArgs e)
  {
    m_sqlCn = new SqlConnection(...);
    //...
  }

  [InState( KnownDB )]
  void WriteTRDetail ()
  {
     m_sqlCn.Open();
     SqlCommand objCommand =
        new SqlCommand("EXEC ...", m_sqlCn);
     SqlDataReader objDataReader =
       objCommand.ExecuteReader();
     // ...
  }
}
```

FIG. 15

1600

```
string GetPersonWebURL (
  [ InReaderState(
    ColumnNames = - "internalurl", "externalurl" ",
    ColumnTypes = - "nchar", "nchar" " ]
  SqlDataReader dr )
 {
  if (dr["internalurl"] = = null)
     if (dr["externalurl"] = = null)
        return "";
     else
      // ...
 }
```

FIG. 16

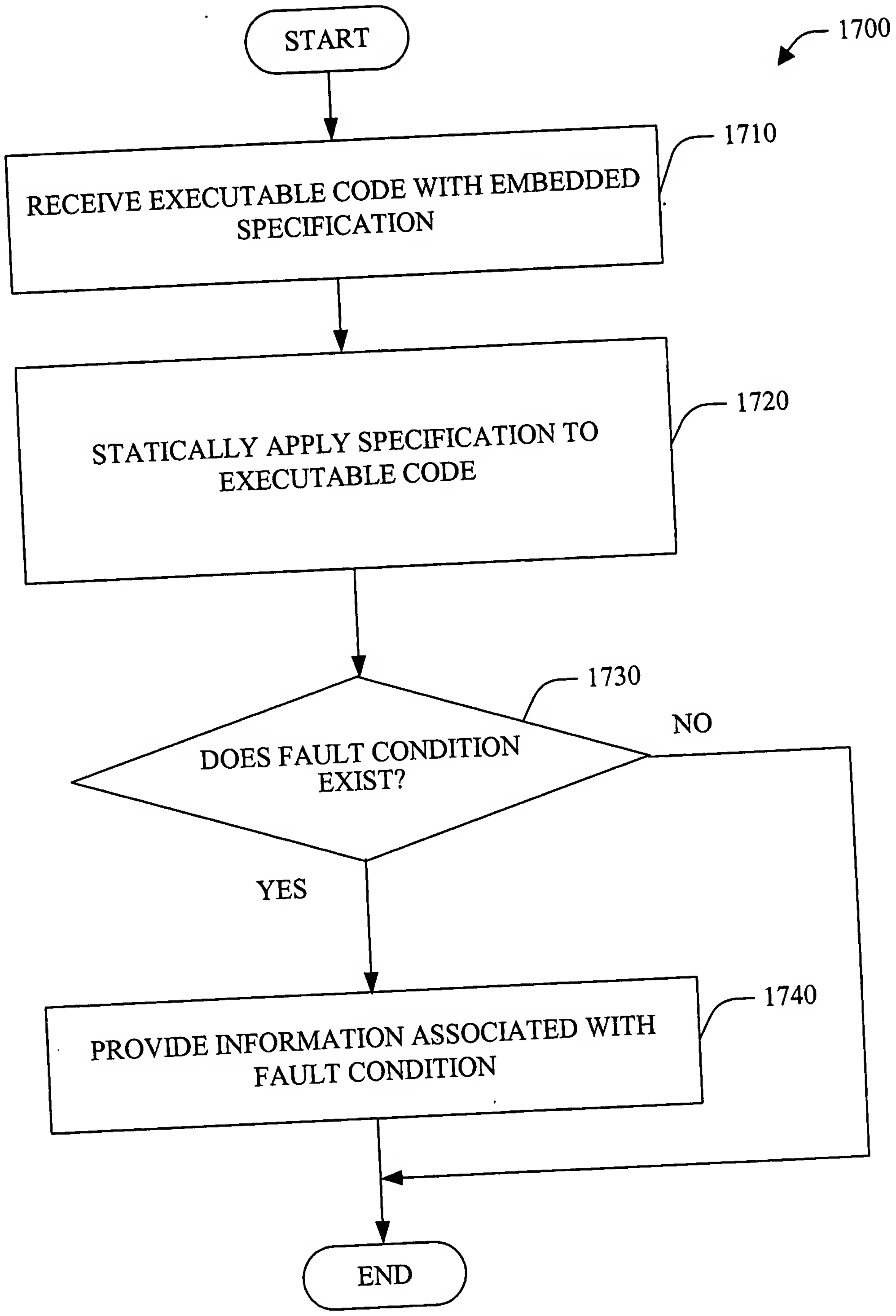
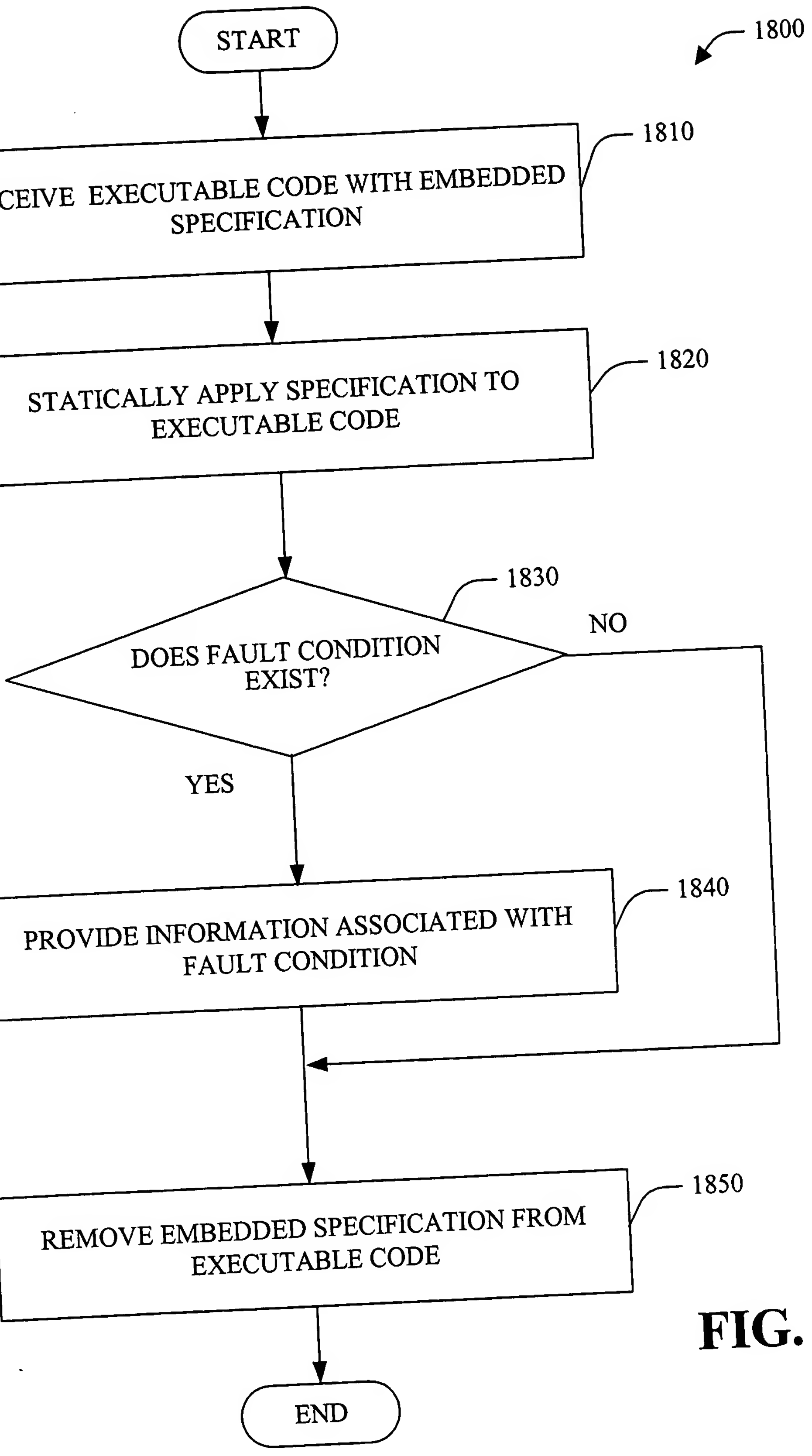
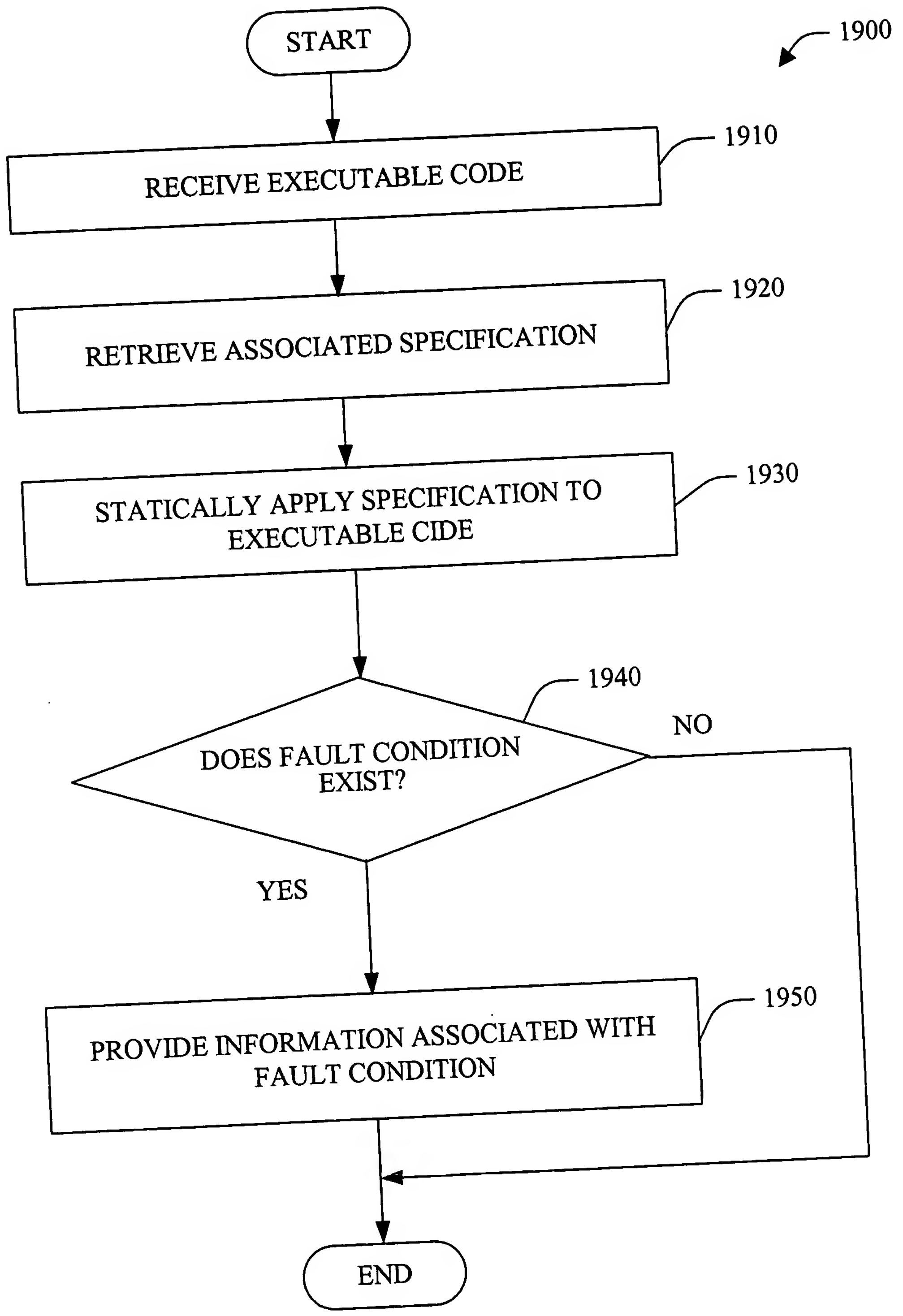

START
1700
RECEIVE EXECUTABLE CODE WITH EMBEDDED SPECIFICATION
1710
STATICALLY APPLY SPECIFICATION TO EXECUTABLE CODE
1720
DOES FAULT CONDITION EXIST?
1730
NO
YES
PROVIDE INFORMATION ASSOCIATED WITH FAULT CONDITION
1740
END

FIG. 17



1800
START
RECEIVE EXECUTABLE CODE WITH EMBEDDED SPECIFICATION
1810
STATICALLY APPLY SPECIFICATION TO EXECUTABLE CODE
1820
1830
DOES FAULT CONDITION EXIST?
NO
YES
PROVIDE INFORMATION ASSOCIATED WITH FAULT CONDITION
1840
REMOVE EMBEDDED SPECIFICATION FROM EXECUTABLE CODE
1850
END

FIG. 18



START
1900
RECEIVE EXECUTABLE CODE
1910
RETRIEVE ASSOCIATED SPECIFICATION
1920
STATICALLY APPLY SPECIFICATION TO EXECUTABLE CIDE
1930
DOES FAULT CONDITION EXIST?
1940
NO
YES
PROVIDE INFORMATION ASSOCIATED WITH FAULT CONDITION
1950
END

FIG. 19

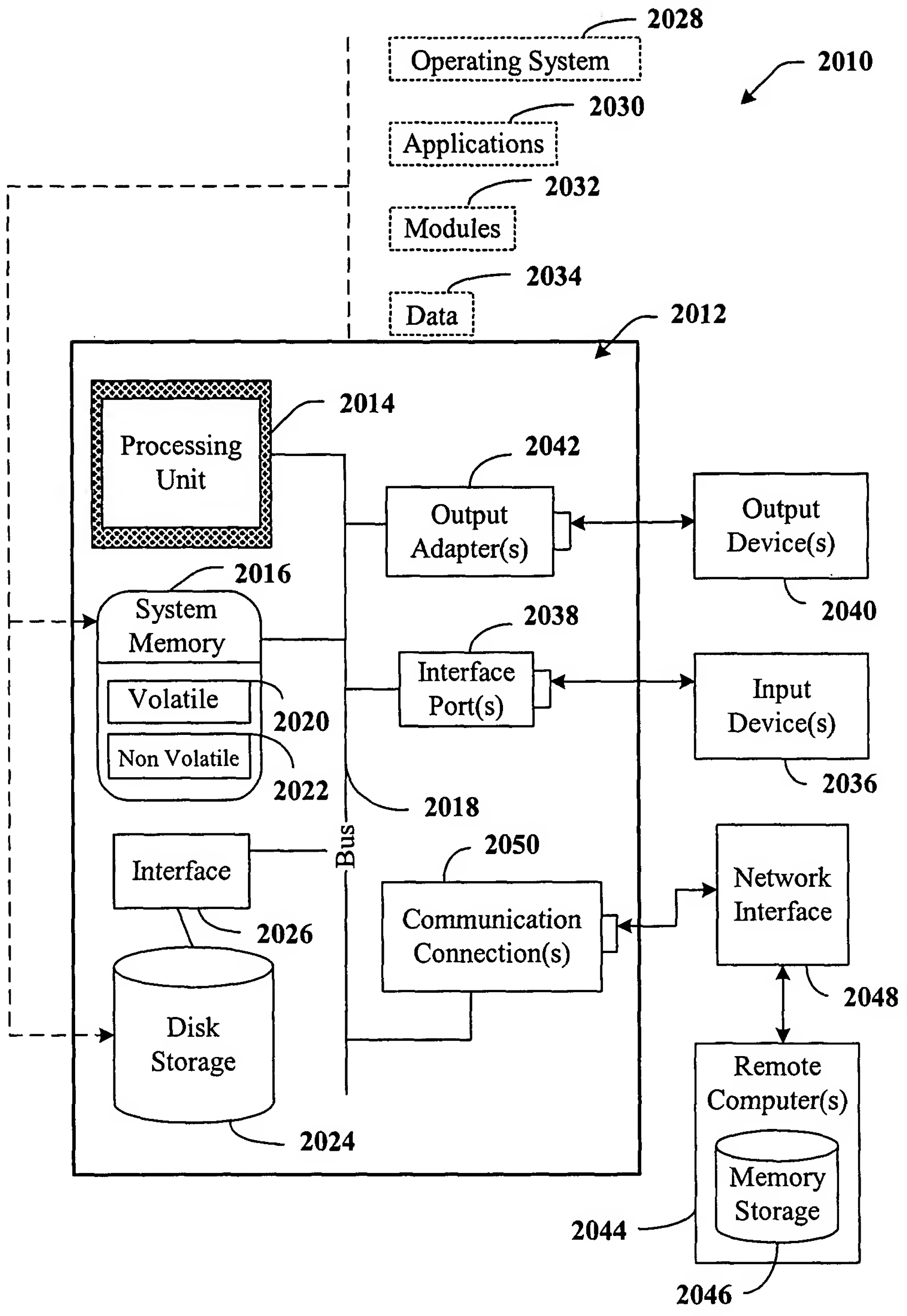

2028
Operating System
2010
2030
Applications
2032
Modules
2034
Data
2012
Processing Unit
2014
2042
Output Adapter(s)
Output Device(s)
2040
2016
System Memory
Volatile
2020
Non Volatile
2022
2038
Interface Port(s)
Input Device(s)
2036
2018
Bus
Interface
2026
2050
Communication Connection(s)
Network Interface
2048
Disk Storage
2024
Remote Computer(s)
Memory Storage
2044
2046

FIG. 20